

## DUAL MODE ARITHMETIC SATURATION PROCESSING

5

### FIELD OF THE INVENTION

The present invention relates to systems and methods in a device such as a processor, including microprocessors and controllers. More particularly, the present invention relates to systems and methods for overflow and saturation processing during accumulator operations.

10

### BACKGROUND OF THE INVENTION

Processors, including microprocessors, digital signal processors and microcontrollers, typically include an accumulator that stores the results of operations performed by the processor. Common operations performed include addition and subtraction. Addition and subtraction operations may cause the result of the operation to exceed the maximum value of the accumulator.

20

In an accumulator, the most significant bit of the accumulator can be used to represent the sign of the number stored in the remaining bits of the accumulator. For example, in a 32 bit accumulator the most significant bit, b31 can represent the sign of the number stored in bits b<sub>30</sub> – b<sub>0</sub>. Using such an arrangement, the accumulator can store a maximum negative number of 0x80000000, where "0x" denotes hexadecimal. The accumulator can store a maximum positive number of 0x7FFFFFFF. Typical saturation processing in the exemplary 32 bit accumulator sets the 32 bit accumulator to the maximum positive number, 0x7FFFFFFF, or the maximum negative number 0x80000000 as the case may require. To illustrate, suppose the following two

25

numbers are added, 0x007FFFF000 and 0x0000001020. The result is 0x0080000020, with an overflow of the result into the sign bit,  $b_{31}$ . Because the result of adding two positive numbers overflowed, the maximum positive number 0x007FFFFFFF is stored in the accumulator.

The above common saturation operation, however, causes the result to be truncated. That is, the actual result of the operation is lost and an approximate result represented by a selected one of the predetermined constants is stored in the accumulator. Thus, the accumulator value after being set by the saturation processing is erroneous. It is desirable to minimize the error introduced by the saturation processing.

## **SUMMARY OF THE INVENTION**

It is an object of the present invention to provide an efficient saturation processing system and method.

It is another object of the present invention to provide a saturation processing system and method that reduces the error introduced by the saturation processing.

It is a further object of the present invention to provide a saturation processing system and method that allows programmers the flexibility of additional accuracy in overflow and saturation processing.

It is still another object of the present invention to provide a saturation processing system and method that provides enhanced computational accuracy in saturation processing of an overflow condition.

It is still a further object of the present invention to provide a saturation processing system and method that can be selectively enabled and disabled.

To achieve the above and other objects, the present invention provides a system for overflow and saturation processing, comprising: an adder, operatively connected to receive first and second operands, and connected to add the operands; an accumulator, operatively connected to store at least a portion of the added operands or at least a portion of a selected one of  
 5 predetermined constants based on control signals; guard bits, operatively connected to store the remaining portion of the added operands or the remaining portion of the selected one of predetermined constants based on the control signals; overflow logic operatively connected to the accumulator and to the guard bits so as to indicate overflow of the accumulator; and saturation logic, operatively connected to the adder, to the guard bits, and connected to provide the control signals based on at least a portion of the added operands at least a portion of the guard bits.

To achieve the above and other object, the present invention also provides a method for overflow and saturation processing in a processor including guard bits and an accumulator, comprising: adding operands to form a result; comparing a portion of the result with a portion of the guard bits; storing either a portion of the result in the accumulator and the remaining portion of the result in the guard bits, or a portion of a selected predetermined constant in the accumulator and the remaining portion of the predetermined constant in the guard bits in accordance with an enable signal and the result of the comparison.

20

### **BRIEF DESCRIPTION OF THE FIGURES**

Figure 1 is a schematic block diagram of a portion of a processor structure that can embody the present invention.

Figure 2 is a schematic block diagram of an exemplary embodiment of the present invention.

## 5 DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

Figure 1 is a schematic block diagram of a portion of a processor structure that can embody the present invention. Referring to Figure 1, saturation logic 20 is coupled to an accumulator 10 and to an adder 30. As will be recognized by those skilled in the art, the adder 30 performs operations on Operand<sub>1</sub> and Operand<sub>2</sub>. In the above example, of adding 0x007FFFF000 and 0x0000001020. The result is 0x0080000020. By comparing the most sign bit, b<sub>31</sub>, of the result with the sign of the accumulator, the overflow condition is detected. The saturation logic 20 causes the maximum positive number 0x007FFFFFFF to be stored in the accumulator 10.

Figure 2 is a schematic block diagram of an exemplary embodiment of the present invention. This is only an example of an embodiment of the present invention. Those skilled in the art will recognize that the logic of Figure 2 can be implemented in a variety of ways, such as, for example, micro code, logic gates, or a programmable logic array. Moreover, while Figure 2 shows a 32-bit accumulator, the invention is not limited to any particular number of bits in an accumulator.

Figure 2 shows an exemplary 32-bit accumulator 60. Bit b<sub>31</sub> of accumulator 60 represents the sign of the value stored in bits b<sub>0</sub> – b<sub>31</sub>. In accordance with the present invention, guard bits 65 are used in conjunction with the accumulator 60. In Figure 2, bits b<sub>39</sub> – b<sub>32</sub> represent the guard bits 65. The present invention is not limited to eight guard bits as shown in Figure 2.

Preferably two or more guard bits are used, with the upper limit of guard bits being determined by the particular application.

NAND gate 70 and OR gate 75 detect an overflow condition of the accumulator. The output of a multiplexer 80 indicates whether the operation performed by adder 90. The output of multiplexer 80 can be applied to a status register, not shown. The state of the overflow bit in the status register can change for each operation performed by adder 90.

Figure 2 shows saturation logic 20 coupled to the guard bits 65, the accumulator 60 and adder 90. The saturation logic 20 provides control signals to selector inputs "A" and "B" of a multiplexer 95. As shown in the illustrative example of Figure 2, the saturation logic 20 compares most significant bits of the guard bits 65 with most significant bits of the result of the operation performed by the adder 90. In Figure 2, AND gates 100 and 105 together with inverters 110 and 115 combine guard bits  $b_{39}$  and  $b_{38}$ . In addition, AND gates 120 and 125 together with inverters 130 and 135 combine bits  $b_{39}$  and  $b_{38}$  of the result of the operation performed by adder 90. AND gates 140 and 145 compare the outputs of And gates 100, 105, 120, and 125 to form control signals that are applied to the A, B inputs of multiplexer 95. AND gates 140 and 145 also receive an Enable signal. In an example embodiment of the present invention, the Enable signal could originate in a mode register that has bits that are set and reset by respective instructions executed by the processor. In the illustrative example shown in Figure 2, when the Enable signal is active, logic 1, the saturation logic allows one of two predetermined constants to be stored in the guard bits 65 and accumulator 60 as indicated by Table 2 shown below. Alternatively, if the Enable signal is inactive, logic 0, the multiplexer/selector 90 allows the result of the operation performed by the adder 90 to be stored in the guard bits 65 and accumulator 60. The Enable signal and the AND gates 140 and 145 function as a means for

providing the control signals in accordance with the Enable signal and in accordance with the comparison of the guard bits 65 and the result of the operation performed by the adder 90.

In addition, together gates 100 – 145 function as a logic means that is responsive to the comparison of the guard bits 65 and the result of the operation performed by the adder 90 so as to selectively provide the control signals so that the accumulator stores at least a portion of the added operands and the guard bits store the remaining portion of the added operands, or the accumulator stores at least a portion of a predetermined constant (e.g., 0x7FFFFFFFFF) and the guard bits store the remaining portion of the predetermined constant (e.g., 0x7FFFFFFFFF).

Table 1 illustrates the logic conditions that give rise to a saturation condition in the illustrative embodiment shown in Figure 2. In Table 1, the "x" denotes a "don't care" condition of the respective bit.

Guard Bit $b_{39}$	Guard Bit $b_{38}$	Result Bit $b_{39}$	Result Bit $b_{38}$	Action
0	1	1	0	Saturation, Store 0x7FFFFFFFFF in Guard bits and Accumulator
1	0	0	1	Saturation, Store 0x800000000 in Guard bits and Accumulator
0	x	0	x	No action, store result of operation performed by Adder
1	x	1	x	No action, store result of operation performed by Adder

**Table 1**

The output of an OR gate 150 indicates if saturation condition has occurred. Typically, the output of the OR gate 150 is applied to a saturation bit in a status register (not shown). It is

common that the saturation bit of the status register be set on the occurrence of saturation and remain set until reset by an instruction executed by the processor. Table 2 below represents logical operation of the multiplexer 95.

The output of the multiplexer 95 is stored in the guard bits 65 and the accumulator 60.

5

<b>A</b>	<b>B</b>	<b>OUT</b>
0	0	Result [ $b_{39} - b_0$ ]
0	1	0x800000000
1	0	0x7FFFFFFFFF
(1	1	Not possible)

**Table 2**

While specific embodiments of the present invention have been illustrated and described, it will be understood by those skilled in the art that changes may be made to those embodiments without departing from the spirit and scope of the invention that is defined by the following claims.